Simulation Infrastructure for Modeling Large Scale Neural Systems

Charles C. Peck, James Kozloski, A. Ravishankar Rao, and Guillermo A. Cecchi

IBM T.J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598
{ cpeck, kozloski, ravirao, gcecchi}@us.ibm.com

Abstract. This paper describes the Large-scale Edge Node Simulator (LENS), a problem solving environment for the implementation of large scale models of neural systems. This work was motivated by the absence of adequate modeling tools for this domain. The object-oriented LENS solution was developed after a rigorous requirements analysis for this class of simulations. An example use of LENS for a complex neural simulation of cortical plasticity is presented. It is shown that LENS is capable of meeting the challenges of simulating large scale models of neural systems.

1 Introduction

This paper describes the Large-scale Edge Node Simulator (LENS), a problem solving environment for the implementation of large scale models of neural systems.

Neural systems are highly complex, structured, and can be viewed from many levels of abstraction. For example, the human brain is composed of roughly 100 billion neurons interconnected by 100 trillion synapses. These fundamental elements are assembled into approximately 850 anatomically identifiable structures [1], such as the cerebellum and the visual cortex, each consisting of specific classes of neurons interconnected in stereotyped patterns. These structures have patterned connections to and from other structures. Finally, these networks of structures interact with the sensors and muscles of the body to generate behavior that is appropriate for a particular environmental context.

Since the earliest experimental studies of neural dynamics [2], simulations have provided value to the understanding of neural systems. Simulations can guide experiments by providing a conceptual framework within which to interpret data, form hypotheses,and allow exploration of phenomena that are not directly accessible. For instance, simulations have been used to model epileptic seizures for clinical purposes [3]. At a higher level of abstraction, large scale simulations can provide a testbed for exploring the fundamental computational principles of global brain function. The value and impact of simulations is expected to increase dramatically with the continuing improvement of computational performance and the full exploitation of recently developed experimental techniques, such as PET and fMRI, multi-electrode recordings from brain areas, and optical recordings.

Adequate tools for large scale neural simulations do not presently exist. Currently available simulation environments are either general purpose or tailored to a specific class of neural models. General purpose tools lack support for neural system-specific modeling, such as simple methods for generating complex network topologies and global architectures. Conversely, environments like Neuron and Genesis provide tools for detailed compartmental modeling of individual neurons with few capabilities for models at other levels of abstraction. Furthermore, the low abstraction level of compartmental modeling makes large-scale simulations of neural systems computationally intractable.

The remainder of the paper presents the specific challenges of large scale neural system modeling and their technological implications, a description of LENS and how LENS solves these challenges, an example application of LENS, and a brief conclusion.

2 Challenges of Large Scale Modeling of Neural Systems

Large scale modeling of neural systems imposes several key technical challenges, including extensibility, scalability, and interoperability. In addition, challenges specific to each phase of the experimental cycle must also be satisfied.

Extensibility is required to enable the introduction and execution of arbitrary models. Furthermore, the simulation must not constrain these models to operate at a specific level of abstraction. As examples, one may desire to model: the visual cortex as a collection of thousands of minicolumn model instances, a minicolumn as a collection of about 100 neuron model instances, or a neuron as a collection of hundreds of compartmental model instances.

Scalability is required to support simulations with complexity spanning many orders of magnitude. For example, a simulation at the level of neurons and synapses might involve a few hundred neurons and thousands of synapses, or it might require $> 10^6$ neurons and $> 10^8$ synapses to model a large structure, such as the hippocampus.

Interoperability of models is required to support the needs of the neuroscientific community. These needs include exchanging and refining models, communicating experimental phenomena, developing higher level models based on models developed by others, and replicating results.

Conducting a complex experiment takes a considerable amount of time from conception through data analysis. While the design and implementation of most simulation systems emphasizes run-time execution performance, experience shows that the time required for setup and initialization of complex simulations exceeds that for run-time execution by many orders of magnitude. The same is true for data collection and analysis. To reduce the time required for a simulation experiment, the simulation environment must therefore expedite each of the three phases of the experimental lifecycle (i.e., setup and initialization, run-time execution, and data collection and analysis). The remainder of this section addresses the specific challenges in each of the three phases.

2.1 Setup and Initialization

To be most efficient, the manner in which the simulation is setup and initialized should be well matched to the manner in which the experiment designer conceptualizes large scale neural systems. In particular, there should be support for instantiating multiple types of models and for establishing complex patterns of connectivity. This latter requirement is especially important for maintaining scalability, as point-to-point specification of connections would not be possible for experiments consisting of large numbers (possibly millions) of interconnected model instances.

The simulation environment must also support the parameterization of the model instances. This challenge is exacerbated by the large numbers of instances, that they may be instantiated from arbitrary models, and that the parameters for each instance may vary or even depend upon the configuration context.

In addition to parameterizing the instances of models, the simulation environment must also support parameterization of connections or relationships among models. For example, the effect one neuron has on another is dependent on where the synapse occurs on the postsynaptic neuron. This position information, which may be parameterized, is not a property of either neuron, rather it is a property of the relationship between them.

2.2 Run-time Execution

Computational efficiency is critical for large scale simulations. The run-time environment must effectively utilize all available computational resources, such as the CPUs and system memory. In addition, the run-time environment should support both time-step and discrete time event update modes to allow the selection of the most appropriate mode for a particular model. For example, since neuron models are continuously and frequently updated they should use the time-step mode. In contrast, models of environmental stimuli may use fewer resources with the higher overhead, but far less frequent, discrete time event updates.

The run-time environment must also support control of computation sequencing at each time step. This control enables integration of heterogeneous models and greater parallelism. Such control would enable, for example, the ability to execute all synapse model instances in parallel prior to executing any neuron model instances. Once all synapse instances have been executed, the neuron model instances could then be executed, also in parallel.

2.3 Data Collection and Analysis

Data collection and analysis is specific to the particular experiment being performed, and must be easy, efficient, involve minimal code development, and maximize the reuse of third party data analysis tools, such as data visualization and statistical analysis packages. For ease and efficiency, data collection and analysis setup should be done in the same way as simulation setup and initialization. Furthermore, it should be possible to extract specific data from individual model instances or desired collections of instances.

The simulation environment must also support a rich set of data collection and analysis operations. For example, it must be possible to perform data collection every time step, when a predicate is satisfied, or when a user interactively indicates. It must also be possible to capture time-series of variables of interest. Finally, the simulation environment should provide interactive access to arbitrary data while a simulation is running.

3 A Large-Scale Neural Simulation Infrastructure

3.1 Overview

This section presents a detailed description of the LENS approach for satisfying each of the requirements for large-scale simulations of neural systems.

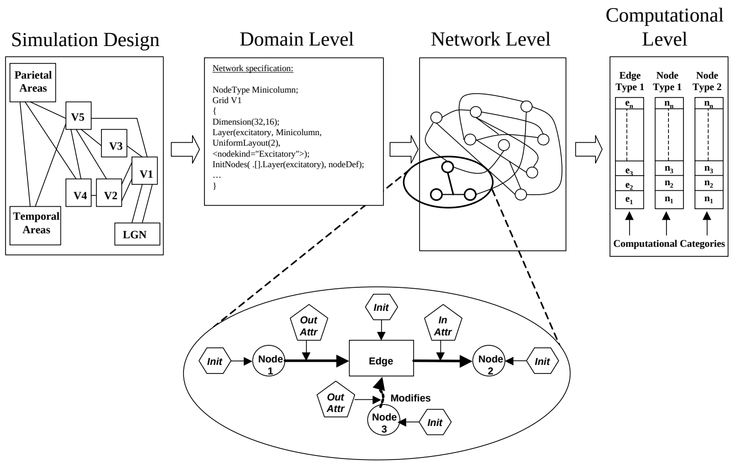
3.2 Modeling Architecture

The primary design philosophy for LENS was to allow experiment designers to express their designs in a manner consistent with their conceptualizations, to allow model developers to write their models for maximum computational efficiency, and to bridge the gap between simulation specification and model development with technology.

This philosophy is illustrated in Figure 1. The “Simulation Design” refers to the experimenter’s conceptualization of the experiment. The “Domain Level” refers to a specification using the terminology and knowledge framework provided by neuroscience. The “Network Level” refers to the technology bridge and the “Computational Level” refers to the level at which model development is performed.

A description of how each of these levels is implemented is given below.

Domain level. To allow the simulation designer to create domain level models without requiring simulation system expertise, LENS includes a specification language. This language allows domain level modeling to proceed using familiar terms, and for domain level simulation designers to access, parameterize, and compose existing models automatically based on a high level specification.



```
Network specification:

NodeType Minicolumn;
Grid V1
{
Dimension(32,16);
Layer(excitatory, Minicolumn,
UniformLayout(2),
<nodekind="Excitatory">);
InitNodes( .[].Layer(excitatory), nodeDef);
…
}
```

Fig. 1. Summary of design philosophy for the problem solving environment.

Network level. To bridge the gap between domain level modeling and its mapping onto the computational infrastructure, the network level generates an isomorphic description of the simulation in the form of an edge-node graph. Both domain level simulation designers and model code developers can address network elements directly through this graph description. The graph is fundamentally constrained in that nodes connect directionally to an arbitrary number of edges while edges connect directionally to precisely two nodes.

To a simulation designer, edges and nodes represent network elements that are invoked within a specification using domain level names (eg., neuron, synapse, minicolumn). To a code developer, these same network elements are computational models that support standard interfaces for initialization, execution, browsing, and data collection and analysis.

Connections within the edge-node graph are traversed through hierarchical collections of node repertoires and edge connection sets. Some aspects of this hierarchy have been proposed previously in other simulation system designs [4]. There are two types of repertoires. Composite repertoires are composed of other repertoires, and grid repertoires consist of a multidimensional array nodes. Grids provide access to nodes according to their grid coordinate system and/or their model type. Grids have arbitrary dimensionality and may support an arbitrary number of nodes at each grid coordinate. Finally, each repertoire provides access to all edges that are entirely contained within it.

Computational level. To provide a basis for model development, LENS provides standard API's for model initialization, parallelization, memory management, execution, browsing, and data collection. These APIs assume a uniprocessor or shared memory multiprocessor system. Model developers can use these APIs to efficiently exploit the available computational resources.

3.3 System Architecture

As shown in Figure 2, the LENS system possesses data stores containing specification files and model libraries, and a simulation infrastructure that includes an initializer and simulation kernel. The initializer consists of a parser for reading specification files and generating the simulation implementation and a loader for dynamically loading the specified models. The simulation kernel owns and executes computational categories, which themselves contain executable models, and the edge-node graph description of the simulation. In addition, a Java-based data browser and simulation control system, and distributed third-party tools for data collection and analysis are depicted running on systems that form dynamic links to the simulation to perform these functions.

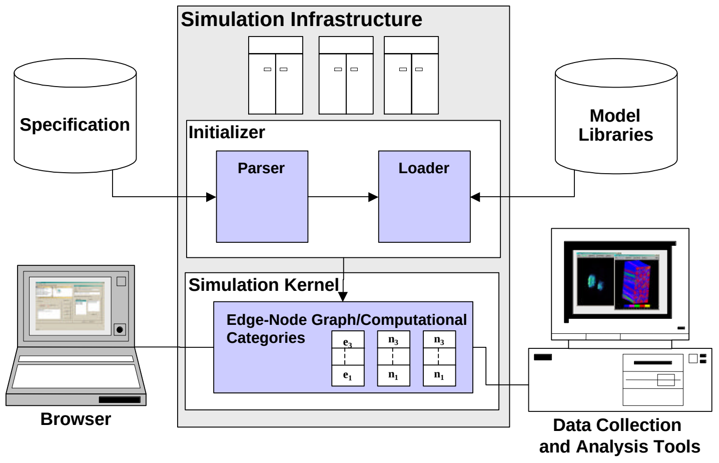


Fig. 2. Architectural design of problem solving environment.

Within this system, data about the simulation originates in the specification files and model libraries. The specification is read by the parser, which, after dynamically loading the specified models and initialization algorithms, generates instances of edges and nodes. These instances are then composed into an edge-node graph using the specified connection algorithms. Nodes and edges, their connectivity, and the relationships between them, are then parameterized algorithmically. Both browsing systems and data collection and analysis tools access the graph through the network level hierarchy.

3.4 Initialization solution

Specification language. The specification language allows the setup and initialization of large-scale simulations of neural systems to proceed using the terms with which the simulation designer conceptualizes these systems. The language is based on a generic programming paradigm, is similar to C++ in its grammar, and uses Standard Template Library syntax. Functor (function objects), variables, and scripts are each specifiable using the language. Multiple types of models can be readily specified for instantiation, then parameterized and connected together in complex patterns using functors. Relationships between models are parameterized separately from the models themselves. The specification language also supports setup and initialization of data collection and analysis.

Loader. For extensibility, all components (eg., nodes, edges, and functors) of the simulation are dynamically loaded from model libraries. The simulation system provides a means to parameterize loaded models easily from within a specification. To support efficient data exchange between arbitrary models, an interface negotiation protocol has been provided that allows the initializer to verify that specified connections are supported, and to establish a direct link between data objects once, prior to simulation run-time.

3.5 Run-time solution and simulation kernel

Thread-pools of CPU-bound kernel threads. Thread pools are available to model code developers as they design and implement computational categories for high performance. Threads are managed by the simulation engine, and bound to all available CPUs in a shared memory system.

Synchronous and asynchronous event queues. To achieve high performance, the simulation engine supports two simulation modes for model computation: time-step and discrete-event. These modes are integrated by the simulation engine through the use of both synchronous and asynchronous event queues. Code developers can therefore choose to implement models using the more efficient mode of operation, and simulation designers can specify the use of these models arbitrarily within a simulation.

Arbitrary phase ordering. The simulation infrastructure allows model code developers to arbitrarily control the sequencing of their model computations through assignment of computational phases during initialization. During implementation, developers assign computational categories their relative computational phases (e.g., *compute edges before nodes*), which are then made absolute when models are loaded during initialization.

3.6 Data Collection and Analysis Solution

The simulation infrastructure creates a easy and efficient means to access, collect, and analyze arbitrary data from within a simulation.

Composable Collection and Analysis Tools. The operations of data collection and analysis are implemented as composable, reusable elements within the simulation infrastructure. Data collection objects are specified, parameterized, and composed using domain modeling terms from within a specification. These objects collect arbitrary data from arbitrary simulation components or collections of components. In addition, the data collection and analysis solution supports an arbitrary, composable triggering scheme, such that data can be collected continuously, over any specified time interval relative to a simulation event, or upon a user command.

Network, Data, and Analysis Browser. Users can access all simulation elements with a Java-based network, data, and analysis browser. The browser allows a user to navigate the edge-node graph, select data for collection and analysis, and invoke arbitrary analysis tools. Tools can be created from prespecified types, or can be dynamically loaded from tool libraries.

Integrated Open Source Data Visualization Tool. Third party, open source data visualization is fully integrated within the simulation infrastructure. The visualization tool, DataExplorer(TM), employs a socket interface to access active data collection and analysis tools within the simulation. Sockets can be created during initialization or dynamically, then directed to DataExplorer(TM) for real-time data display. Because data collection and analysis tools can access arbitrary data, the entire simulation is accessible for visualization.

4 Simulation example

4.1 Rationale and simulation overview

In order to validate our simulation environment, we implemented a previously published neural simulation [5], which modeled results from experiments on the plasticity of connections in somatosensory cortex [6].

The somatosensory cortex is organized into regions, such that each hand digit maps to a specific region. Neurons within each region are organized into tight clusters called neuronal groups, such that synaptic weights between neurons within a neuronal group is significantly higher than weights between neurons in different groups. This group structure is hypothesized to arise as a consequence of self-organization through synaptic changes driven by the sensory input.

In our simulation, the somatosensory cortex receives inputs from four digits. The digits were modeled using two receptor sheets, derived from a grid of size 32x16. Each receptor sheet projects to a patch of cortex, overlaid on a grid of size 32x16. The simulation includes three classes of neurons and two classes of synaptic connections, for a total of 1500 neurons and 170,000 synaptic connections. The simulation also includes rules for synaptic modification, and specific patterns of anatomical connectivity between the network elements. The simulation was run on a 4-processor IBM RS/6000 machine running AIX Version 4.3.2.

LENS was integrated with a third party data analysis tool, the IBM DataExplorer(TM) data visualization system.

4.2 Specification example

The simulation was implemented in our environment using the specification language in about 50 lines. The following excerpt shows the ability of the language to capture complex configurations in a compact way.

Cortical simulation specification excerpt:

```
 1  NodeType PnExCell;
 2  NodeType PnInCell;
 3  Grid Cortex
 4  {
 5      Dimension(32,16);
 6      Layer(excitatory, PnExCell, UniformLayout(2), < nodekind="Excitatory" >);
 7      InitNodes(.[].Layer(excitatory), nodeDef);
 8      Layer( inhibitory, PnInCell, UniformLayout(1), < nodekind="Inhibitory" >);
 9      InitNodes(.[].Layer(inhibitory), nodeDef);
10      list<int> e2e = {1,8,8};
11      list<int> e2i = {1,4,8,24,32};
12      list<int> i2e = {1,8,8};
13      list<int> i2i = {0};
14      centerSurround(.[],.[], e2e, e2i, i2e, i2i);
15  };
```

In this specification, two node types are declared “PnExCell” and “PnInCell” (ll.1-2). Next a grid repertoire “Cortex” is declared (l.3), its size and dimensionality specified (l.5), and its layers declared and initialized (ll.6-9). Finally, a functor, “centerSurround,” is employed that connects the Cortex’s own “Excitatory” and “Inhibitory” layers (specified as “.[]” in the centerSurround’s first two arguments) using a geometric pattern specified in four list arguments (specified in ll.10-14).

4.3 Simulation results

Figure 3 shows the output from the DataExplorer(TM) module of our simulation after running this experiment. The left panel shows the pattern of neural activation in response to a single stimulus in both the inhibitory (leftmost) and excitatory (rightmost) layers of the cortical sheet model. The right panel shows the connections from the receptor sheet to the somatosensory cortex, after repeated stimulation of the digits. The synaptic strength of the connections is color-coded. The emergence of neuronal groups can be observed as clusters of high synaptic strength, in agreement with previous results [15].

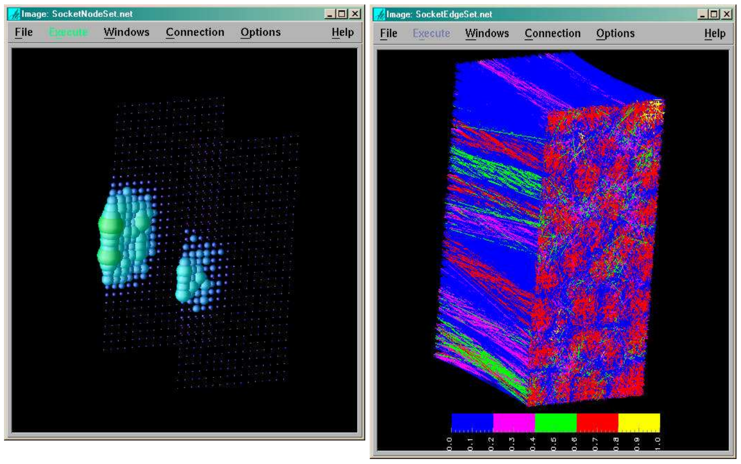


Image: SocketNodeSet.net
File Execute Windows Connection Options Help
Image: SocketEdgeSet.net
File Execute Windows Connection Options Help
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0

Fig. 3. Results of test simulation visualized with DataExplorer(TM).

5 Conclusion

This paper was motivated by the lack of adequate problem solving environments for implementing large scale simulations of neural systems. Consequently, this paper has identified the challenges associated with meeting the needs of this domain. Based on these challenges, the LENS simulation environment was designed and implemented. It has been demonstrated that LENS is able to implement, execute, and analyze large scale, complex simulations of neural systems. Finally, the has shown that it is possible to meet the challenges of large scale simulations of neural systems and that LENS has done so.

References

1. See http://rprcsgi.rprc.washington.edu/neuronames/index1.html
2. Hodgkin, A.L., Huxley, A.F.: A quantitative description of membrane current and its application to conduction and excitation in nerve. J. Physiol. 117:500-544 (1952).
3. Schiff, S.J.: Forecasting brain storms, Nat. Med. 4(10):1117-8 (1998).
4. Reeke, G.N., Edelman, G. M.: Selective Networks and Recognition Automata, Annals New York Academy of Sciences. 426:181-201 (1984).
5. Pearson, J.C., Finkel, L.H., Edelman, G.H.: Plasticity in the organization of adult cerebral cortical maps: a computer simulation based on neuronal group selection, J. Neuroscience, 7(12): 4209-4233, December 1987.
6. Merzenich, M.M., Kaas, J.H., Wall, J.T., Neson, R.J., Sur, M., Felleman, D.J.: Topographic reorganization of somatosensory cortical ares 3b and 1 in adult monkeys following restricted deafferentation, Neuroscience 10:639-665, 1983.